Spring Cloud

==========

Spring Cloud is a Spring module that provides the RAD (Rapid Application Development) feature to the Spring framework.

Spring Cloud is a framework for building robust cloud applications. Spring Cloud provides a solution to the commonly encountered patterns when developing a distributed system.

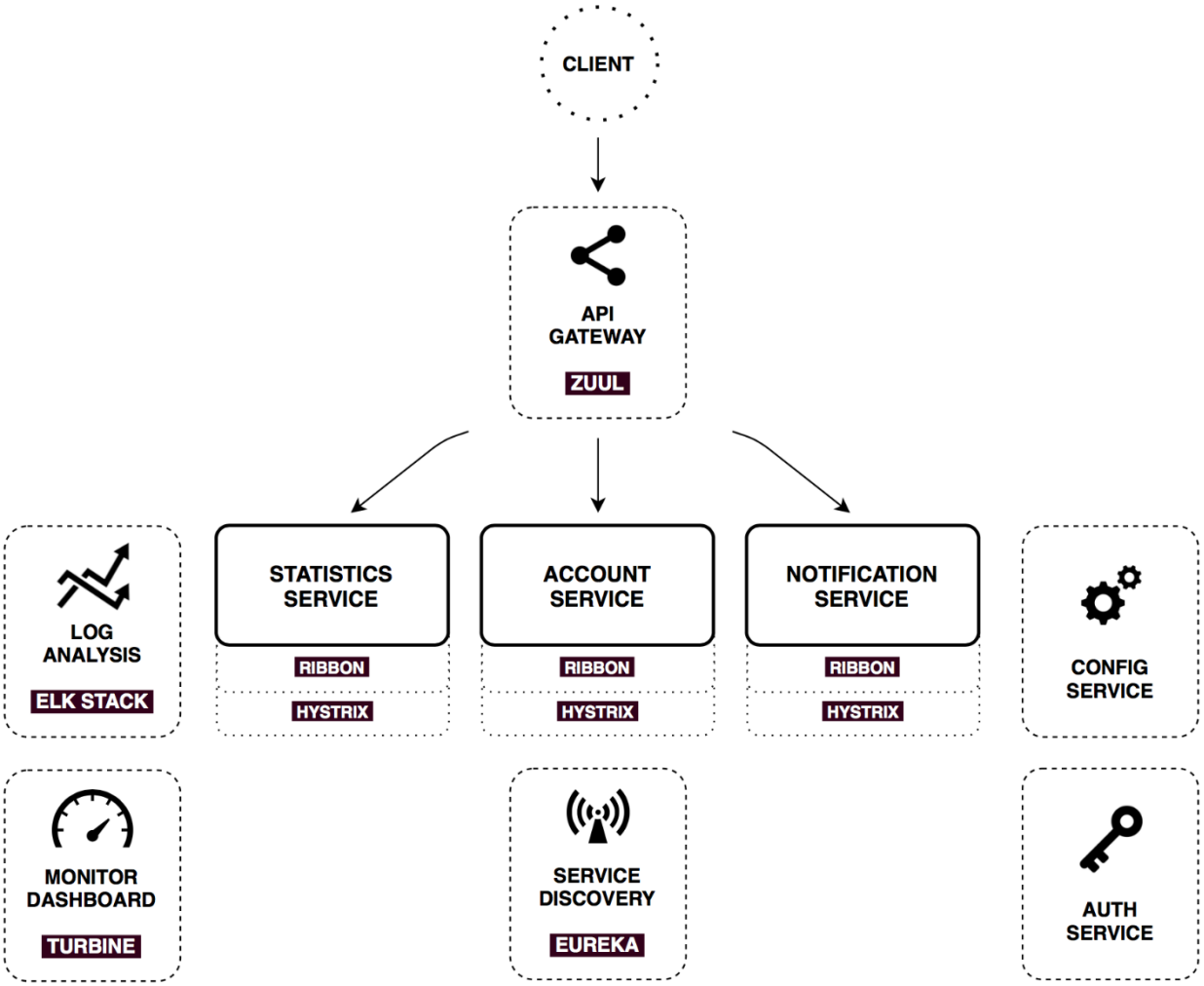
Why Spring Cloud:

==========

Spring Cloud framework provides tools for developers to build a robust cloud application quickly. We can also build the microservice-based applications.

For example, configuration management, service discovery, circuit breakers, intelligent routing.

Spring Cloud Tools:



Spring Cloud provides tools for developers to build some of the common patterns in distributed systems quickly.

Exmple:

****

1.Configuration Management

2.Service Discovery

3.Circuit-Breakers

4.Routing

5.Cloud Bus

6.Load Balancing

Cloud Foundary:

==========

Cloud Foundry is an open-source, multi-cloud Platform as a Service (PaaS). You can deploy your application on your own computing infrastructure.

Features of Spring Cloud:

==========

The great part about Spring Cloud is that it builds the concept of Spring Boot. Spring cloud is built upon some of the common building blocks of Spring framework which are as follows:

1.Configuration Management

2.Service Discovery

3.Circuit-Breakers

4.Routing

5.Cloud Bus

6.Load Balancing

7.Service to Serivce Calls

8.Ditributed Tracing

9.API Gate Way

Distributed Messaging

Configuration Management:

==========

Spring Cloud configuration components provide server-side and client-side support for externalized configuration in a distributed system. We can manage the external properties with config server for applications across all environments. Spring Cloud config server can use Git, SVN (Apache Subversion), filesystem, and Vault to Store config.

Config clients (microservice app) retrieve the configuration client from the server on startup.

Service Discovery:

==========

The service discovery is the automatic detection of devices and services over the network. In other words, service discovery is how an application and microservices connect in the distributed environment.

There are two discovery patterns: Client-side discovery and Server-side discovery.

Client-side discovery:

==========

In the Client-side discovery, client is responsible for determining the network location of available services. The client uses a load-balancing algorithm to select one of the available services and make a request. Netflix OSS is an example of a client-side discovery pattern.

Server-side discovery:

==========

In the server-side discovery, the client makes an HTTP request to a service through a load balancer. The load balancer contacts to service registry and route each request to an available service instance. Similar to client-side discovery, service instances are registered and deregistered with the service registry. The AWS ELB (Elastic Load Balancer) is an example of server-side discovery. ELB balances the external traffic from the internet.

Circuit Breakers:

==========

Netflix has created a library called Hystrix. It implements the circuit breakers pattern. Circuit breakers calculate when to open and close the circuit and what to do in case of failure. When all services fail at some point, the circuit breaker handles these failures gracefully. The circuit breakers have three states: OPEN, CLOSED, and HALF-OPEN State.

CLOSED State:

==========

Apllication flow is working like below. CLient calling the supplier in middle Circuit-Breaker sits,

Client ==> Circuit-Breaker ==> Supplier Application

In the Above flow if Circuit-Breaker is in CLosed state all calls pass through to the supplier microservices. It responds without any latency.

OPEN State:

==========

Apllication flow is working like below. CLient calling the supplier in middle Circuit-Breaker sits,

Client ==> Circuit-Breaker ==> Supplier Application

The circuit breaker returns an error calls without executing the supplier api.

HALF-OPEN State:

==========

Apllication flow is working like below. CLient calling the supplier in middle Circuit-Breaker sits,

Client ==> Circuit-Breaker ==> Supplier Application

The circuit turns to HALF-OPEN state when a function execution is timed out. It test that underlying problem still exists or not. It is a monitoring and feedback mechanism. It makes a trial call to supplier microservices to check if it has recovered. If the call to the supplier is timed out, then the circuit remains in the OPEN state. If the call return success, the circuit-switched to the CLOSED state. The circuit breaker returns all external calls to the service with an error during the HALF-OPEN State.

Routing:

==========

The cloud application made up of many microservices so the communication will be critical. Spring Cloud supports communication via messaging or HTTP request. Routing uses Netflix Ribbon and Open Feign.

API GateWay:

==========

API Gateway allows us to route API request (external or internal) to connect services. It also provides a library for building an API gateway on the top of Spring MVC. Its aims to provide cross-cutting concerns to them, such as security and monitoring

Features of API Gateway

==========

Built on Spring framework 5, project reactor and Spring Boot 2.0

Able to match routes on any requested attribute

Predicates and filters are specific to routes

Hystrix circuit Breaker integration

Spring Cloud Discovery Client integration

Easy to write Predicates and filters

Request Rate Limiting

Path rewriting

Tracing:

==========

Spring Cloud's other functionality is distributed tracing. Tracing is a single request to get data from the application. Tracing results in an exponentially larger number of requests to various microservices.

We can add Spring Cloud Sleuth library in our project to enable tracing. Sleuth is responsible for recording timing, which is used for latency analysis. We can export this timing to Zipkin.